1 Branch Prediction [15 points]

Consider the following high level language code segment:

```c
int array[1000] = { /* random values */ };  
int sum1 = 0, sum2 = 0, sum3 = 0, sum4 = 0; 

for (i = 0; i < 1000; i ++)  
{
    if (i % 4 == 0)  
        sum1 += array[i];  
    else 
        sum2 += array[i]; 

    if (i % 2 == 0) 
        sum3 += array[i]; 
    else 
        sum4 += array[i];  
}
```

(a) What is the prediction accuracy for each of the three branches using a per-branch last-time predictor (assume that every per-branch counter starts at “not-taken”)? Please show all of your work.

(b) What is the prediction accuracy for each of the three branches when a per-branch 2-bit saturating counter-based predictor is used (assume that every per-branch counter starts at “strongly not-taken”)? Please show all of your work.

(c) What is the prediction accuracy for both Branch 2 and Branch 3, when the counter starts at (i) “weakly not-taken” and (ii) “weakly taken”?

(d) What is the prediction accuracy for each of the three branches when a two-level global branch predictor with a two-bit global history register and a separate pattern history table per branch, consisting of 2-bit saturating counters for every entry, is used? Assume that both bits of the global history register are initialized to “not-taken” and the 2-bit saturating counters in the pattern history tables are initialized to “strongly not-taken”. When calculating prediction accuracy, ignore the first 500 loop iterations.

2 Branch Prediction [20 points]

Suppose we have the following loop executing on a pipelined LC-3b machine.

```assembly
DOIT
STW R1 ← R6, #0
ADD R6 ← R6, #2
AND R3 ← R1, R2
BRz EVEN
ADD R1 ← R1, #3
ADD R5 ← R5, #1
```
BRp DOIT
EVEN ADD R1 ← R1, #1
ADD R7 ← R7, #-1
BRp DOIT

Assume that before the loop starts, the registers have the following decimal values stored in them:

<table>
<thead>
<tr>
<th>Register</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>R0</td>
<td>0</td>
</tr>
<tr>
<td>R1</td>
<td>0</td>
</tr>
<tr>
<td>R2</td>
<td>1</td>
</tr>
<tr>
<td>R3</td>
<td>0</td>
</tr>
<tr>
<td>R4</td>
<td>0</td>
</tr>
<tr>
<td>R5</td>
<td>5</td>
</tr>
<tr>
<td>R6</td>
<td>4000</td>
</tr>
<tr>
<td>R7</td>
<td>5</td>
</tr>
</tbody>
</table>

The fetch stage takes one cycle, the decode stage also takes one cycle, the execute stage takes a variable number of cycles depending on the type of instruction (see below), and the store stage takes one cycle.

All execution units (including the load/store unit) are fully pipelined and the following instructions that use these units take the indicated number of cycles:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Number of Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>STW</td>
<td>3</td>
</tr>
<tr>
<td>ADD</td>
<td>2</td>
</tr>
<tr>
<td>AND</td>
<td>3</td>
</tr>
<tr>
<td>BR</td>
<td>1</td>
</tr>
</tbody>
</table>

Data forwarding is used wherever possible. Instructions that are dependent on the previous instructions can make use of the results produced right after the previous instruction finishes the execute stage.

The target instruction after a branch can be fetched when the BR instruction is in ST stage. For example, the execution of an ADD instruction followed by a BR would look like:

ADD F | D | E1 | E2 | E3 | ST
BR F | D | - | - | - | E1 | ST
TARGET F | D

A scoreboard mechanism is used.

Answer the following questions:

1. How many cycles does the above loop take to execute if no branch prediction is used (the pipeline stalls on fetching a branch instruction, until it is resolved)?
2. How many cycles does the above loop take to execute if all branches are predicted with 100% accuracy?
3. How many cycles does the above loop take to execute if a static BTFN (backward taken-forward not taken) branch prediction scheme is used to predict branch directions? What is the overall branch prediction accuracy? What is the prediction accuracy for each branch?
3 Scoreboarding [10 points]

Conventional scoreboarding as discussed in class sometimes introduces stalls when they might not actually be necessary. Consider the following code sequence:

ADD R2 ← R1, R3
ADD R2 ← R4, R5

Answer the following questions:

1. Why would the second instruction in this sequence stall in a conventional scoreboarding processor? (Hint: what can’t the scoreboarding table track?)

2. Propose a scoreboard-based dependency check mechanism that addresses this issue. Show the structure of the new scoreboard table clearly, show the conditions for stalling an instruction (and in which stage this stall occurs) due to dependences, and show how the table is updated by instructions as they pass through the pipeline stages.

4 Interference in Two-Level Branch Predictors [20 points]

Assume a two-level global predictor with a global history register and a single pattern history table shared by all branches (call this “predictor A”).

1. We call the notion of different branches mapping to the same locations in a branch predictor ”branch interference”. Where do different branches interfere with each other in these structures?

2. Compared to a two-level global predictor with a global history register and a separate pattern history table for each branch (call this “predictor B”),
   (a) When does predictor A yield lower prediction accuracy than predictor B? Explain. Give a concrete example. If you wish, you can write source code to demonstrate a case where predictor A has lower accuracy than predictor B.
   (b) Could predictor A yield higher prediction accuracy than predictor B? Explain how. Give a concrete example. If you wish, you can write source code to demonstrate this case.
   (c) Is there a case where branch interference in predictor structures does not impact prediction accuracy? Explain. Give a concrete example. If you wish, you can write source code to demonstrate this case as well.

5 Branch Prediction vs Predication [30 points]

Consider two machines A and B with 15-stage pipelines with the following stages.

- Fetch (one stage)
- Decode (eight stages)
- Execute (five stages).
- Write-back (one stage).

Both machines do full data forwarding on flow dependences. Flow dependences are detected in the last stage of decode and instructions are stalled in the last stage of decode on detection of a flow dependence.

Machine A has a branch predictor that has a prediction accuracy of P%. The branch direction/target is resolved in the last stage of execute.

Machine B employs predicated execution, similar to what we saw in lecture.

1. Consider the following code segment executing on Machine A:
add  r3 ← r1, r2
sub  r5 ← r6, r7
beq  r3, r5, X
addi r10 ← r1, 5
add  r12 ← r7, r2
add  r1 ← r11, r9
X: addi r15 ← r2, 10
.....

When converted to predicated code on machine B, it looks like this:

add  r3 ← r1, r2
sub  r5 ← r6, r7
cmp  r3, r5
addi.ne r10 ← r1, 5
add.ne r12 ← r7, r2
add.ne r14 ← r11, r9
add  r15 ← r2, 10
.....

(Assume that the condition codes are set by the “cmp” instruction and used by each predicated “.ne” instruction. Condition codes are evaluated in the last stage of execute and can be forwarded like any other data value.)

This segment is repeated several hundreds of times in the code. The branch is taken 40% of the time and not taken 60% of the time. On an average, for what range of P would you expect machine A to have a higher instruction throughput than machine B?

2. Consider another code segment executing on Machine A:

add  r3 ← r1, r2
sub  r5 ← r6, r7
beq  r3, r5, X
addi r10 ← r1, 5
add  r12 ← r10, r2
add  r14 ← r12, r9
X: addi r15 ← r14, 10
.....

When converted to predicated code on machine B, it looks like this:

add  r3 ← r1, r2
sub  r5 ← r6, r7
cmp  r3, r5
addi.ne r10 ← r1, 5
add.ne r12 ← r10, r2
add.ne r14 ← r12, r9
add  r15 ← r14, 10
.....

(Assume that the condition codes are set by the “cmp” instruction and used by each predicated “.ne” instruction. Condition codes are evaluated in the last stage of execute and can be forwarded like any other data value.)

This segment is repeated several hundreds of times in the code. The branch is taken 40% of the time and not taken 60% of the time. On an average, for what range of P would you expect machine A to have a higher instruction throughput than machine B?
### 6 Out-of-order Execution [30 points]

A five instruction sequence executes according to Tomasulo’s algorithm. Each instruction is of the form ADD DR,SR1,SR2 or MUL DR,SR1,SR2. ADDs are pipelined and take 9 cycles (F-D-E1-E2-E3-E4-E5-WB). MULs are also pipelined and take 11 cycles (two extra execute stages). An instruction must wait until a result is in a register before it sources it (reads it as a source operand). For instance, if instruction 2 has a read-after-write dependence on instruction 1, instruction 2 can start executing in the next cycle after instruction 1 writes back (shown below).

<table>
<thead>
<tr>
<th>Instruction 1</th>
<th>F</th>
<th>D</th>
<th>E1</th>
<th>E2</th>
<th>E3</th>
<th>...</th>
<th>WB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instruction 2</td>
<td>F</td>
<td>D</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>E1</td>
<td>-</td>
</tr>
</tbody>
</table>

The machine can fetch one instruction per cycle, and can decode one instruction per cycle.

The register file before and after the sequence are shown below.

<table>
<thead>
<tr>
<th>Valid</th>
<th>Tag</th>
<th>Value</th>
<th>Valid</th>
<th>Tag</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>R0</td>
<td>1</td>
<td>4</td>
<td>R0</td>
<td>1</td>
<td>310</td>
</tr>
<tr>
<td>R1</td>
<td>1</td>
<td>5</td>
<td>R1</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>R2</td>
<td>1</td>
<td>6</td>
<td>R2</td>
<td>1</td>
<td>410</td>
</tr>
<tr>
<td>R3</td>
<td>1</td>
<td>7</td>
<td>R3</td>
<td>1</td>
<td>31</td>
</tr>
<tr>
<td>R4</td>
<td>1</td>
<td>8</td>
<td>R4</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>R5</td>
<td>1</td>
<td>9</td>
<td>R5</td>
<td>1</td>
<td>9</td>
</tr>
<tr>
<td>R6</td>
<td>1</td>
<td>10</td>
<td>R6</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>R7</td>
<td>1</td>
<td>11</td>
<td>R7</td>
<td>1</td>
<td>21</td>
</tr>
</tbody>
</table>

(a) Before  (b) After

(a) Complete the five instruction sequence in program order in the space below. Note that we have helped you by giving you the opcode and two source operand addresses for the fourth instruction. (The program sequence is unique.)

Give instructions in the following format: “opcode destination ← source1, source2.”

```
            ←           ,
            ←           ,
            ←           ,
            ←           ,
MUL          ← R6 ,  R6
            ←           ,
```

(b) In each cycle, a single instruction is fetched and a single instruction is decoded.

Assume the reservation stations are all initially empty. Put each instruction into the next available reservation station. For example, the first ADD goes into “a”. The first MUL goes into “x”. Instructions remain in the reservation stations until they are completed. Show the state of the reservation stations at the end of cycle 8.

Note: to make it easier for the grader, when allocating source registers to reservation stations, please always have the higher numbered register be assigned to source2.
(c) Show the state of the Register Alias Table (Valid, Tag, Value) at the end of cycle 8.

<table>
<thead>
<tr>
<th>Valid</th>
<th>Tag</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>R0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R7</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The following data flow graph requires three non-negative integer input values 1 (one), N, k as shown. It produces a value Answer, also shown.

What function does this data flow graph perform (in less than ten words, please)?